

ScaLAPACK

Introduction to Scalapack

Timothy H. Kaiser, Ph.D.

tkaiser@sdsc.edu

Thanks to...

Kent Milfeld who wrote most of the text
Laura Nett who helped and wrote the
decomposition routines

Description

- A Scalable Linear Algebra Package (ScaLAPACK)
- A library of linear algebra routines for MPP and NOW machines.
- Problem Solvers
 - Linear Equations
 - Least Squares
 - Eigenvalue

Technical Information

- Technical Working Notes
 - <http://www.netlib.org/lapack/lawns/>

Attributes

- Coding Emphasis
 - Scalability
 - Efficiency
 - Reliability (error bounds)
 - Portability
 - Flexibility
 - Ease of use

Support

- Support in Commercial Packages
 - NAG Parallel Library
 - IBM ESSL
 - CRAY Scientific Library
 - VNI IMSL
 - Fujitsu, HP/Convex, Hitachi, NEC

Components

- ScaLAPACK Components
 - LAPACK (Linear Algebra PACKAGE)
 - BLAS (Basic Linear Algebra Subroutines)
 - PBLAS (Parallel BLAS, a distributed memory version of BLAS)
 - BLACS (Basic Linear Algebra Communications Subroutines)

Packages

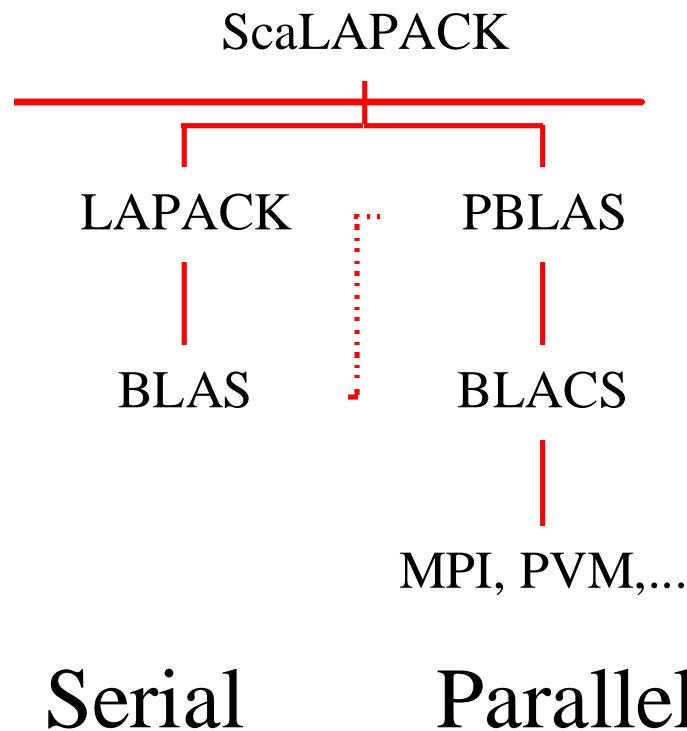
- LAPACK
 - Routines are single-PE Linear Algebra solvers, utilities, etc.
- BLAS
 - Single-PE processor work-horse routines (vector, vector-matrix & matrix-matrix)
- PBLAS
 - Parallel counterparts of BLAS. Relies on BLACS for blocking and moving data.

Packages (cont.)

- BLACS

- Constructs 1-D & 2-D processor grids for matrix decomposition and nearest neighbor communication patterns. Uses message passing libraries (MPI, PVM, MPL, NX, etc) for data movement.

Dependencies & Parallelism of Component Packages



Data Partitioning

- Initialize BLACS routines
 - Set up 2-D mesh (mp, np)
 - Set (CorR) column or Row Major Order
 - Call returns context handle (icon)
 - Get row & column through gridinfo
(myrow,mycol)

```
blacs_gridinit( icon, CorR, mp, np )
```

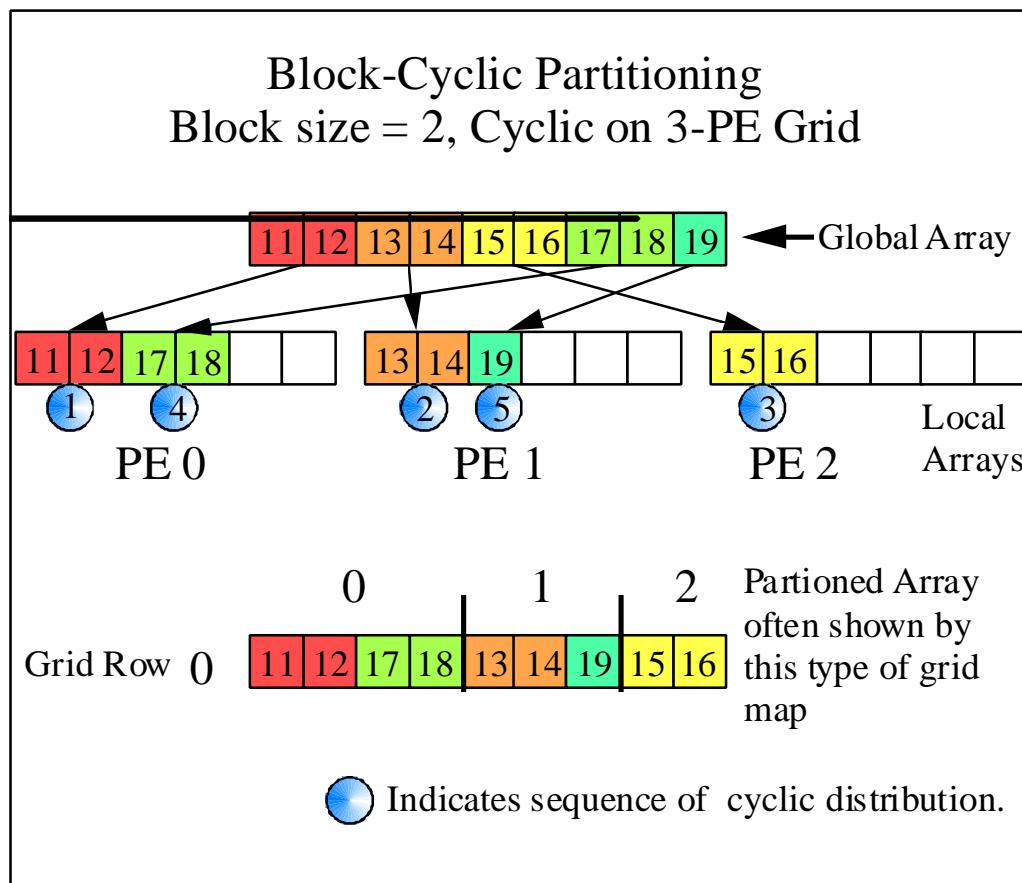
```
blacs_gridinfo( icon, mp_info, np_info, myrow, mycol)
```

Data Partitioning (cont.)

- Block-cyclic Array Distribution.
 - Block: (groups of sequential elements)
 - Cyclic: (round-robin assignment to PEs)

- Example 1-D Partitioning:
 - Global 9-element array distributed on a 3-PE grid with 2 element blocks: block(2)-cyclic(3)

Block-Cyclic Partitioning



2-D Partitioning

- Example 2-D Partitioning
 - $A(9,9)$ mapped onto PE(2,3) grid block(2,2)
 - Use 1-D example to distribute column elements in each row, a block(2)-cyclic(3) distribution.
 - Assign rows to first PE dimension by block(2)-cyclic(2) distribution.

11	12	13	14	15	16	17	18	19
21	22	23	24	25	26	27	28	29
31	32	33	34	35	36	37	38	39
41	42	43	44	45	46	47	48	49
51	52	53	54	55	56	57	58	59
61	62	63	64	65	66	67	68	69
71	72	73	74	75	76	77	78	79
81	82	83	84	85	86	87	88	89
91	92	93	94	95	96	97	98	99

Global Matrix

Global Matrix (9x9)
Block Size =2x2
Cyclic on 2x3 PE Grid

11	12	17	18					
21	22	27	28					
51	52	57	58					
61	62	67	68					
91	92	97	98					

PE (0,0)

13	14	19						
23	24	29						
53	54	59						
63	64	69						
93	94	99						

PE (0,1)

15	16							
25	26							
55	56							
65	66							
95	96							

PE (0,2)

31	32	37	38					
41	42	47	48					
71	72	77	78					
81	82	87	88					

PE (1,0)

33	34	39						
43	44	49						
73	74	79						
83	84	89						

PE (1,1)

35	36							
45	46							
75	76							
85	86							

PE (1,2)

11	12	17	18		
21	22	27	28		
51	52	57	58		
61	62	67	68		
91	92	97	98		

PE (0,0)

13	14	19			
23	24	29			
53	54	59			
63	64	69			
93	94	99			

PE (0,1)

15	16				
25	26				
55	56				
65	66				
95	96				

PE (0,2)

31	32	37	38		
41	42	47	48		
71	72	77	78		
81	82	87	88		

PE (1,0)

33	34	39			
43	44	49			
73	74	79			
83	84	89			

PE (1,1)

35	36				
45	46				
75	76				
85	86				

PE (1,2)

2 x 3 Processor Grid

0	1	2			
11	12	17	18	13	14
21	22	27	28	23	24
51	52	57	58	53	54
61	62	67	68	63	64
91	92	97	98	93	94

0	1	2			
31	32	37	38	33	34
41	42	47	48	43	44
71	72	77	78	73	74
81	82	87	88	83	84

Partitioned Array

Syntax for descinit

`descinit(idesc, m,n, mb,nb, i,j, icon, mla, ier)`

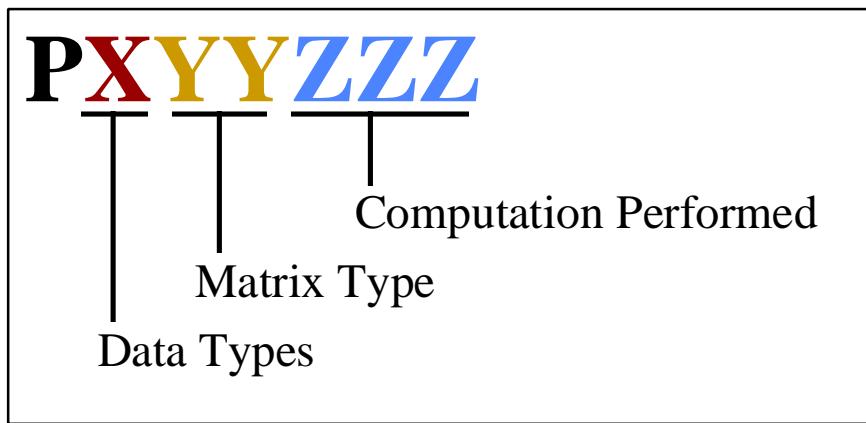
IorO	arg	Description
OUT	<code>idesc</code>	Descriptor
IN	<code>m</code>	Row Size (Global)
IN	<code>n</code>	Column Size (Global)
IN	<code>mb</code>	Row Block Size
IN	<code>nb</code>	Column Size
IN	<code>i</code>	Starting Grid Row
IN	<code>j</code>	Starting Grid Column
IN	<code>icon</code>	BLACS context
IN	<code>mla</code>	Leading Dimension of Local Matrix
OUT	<code>ier</code>	Error number

The starting grid location is usually ($i=0, j=0$).

API (Application Program Interfaces)

- Drivers
 - Solves a Complete Problem
- Computational Components
 - Performs Tasks: LU factorization, etc.
- Auxiliary Routines
 - Scaling, Matrix Norm, etc.
- Matrix Redistribution/Copy Routine
 - Matrix on PE grid1 -> Matrix on PE grid2

- Prepend LAPACK equivalent names with P.



Data Type	real	double	cmplx	dble cmplx
X	S	D	C	Z

Matrix Types (YY)

XXYYZZZ

- DB** General Band (diagonally dominant-like)
- DT** general Tridiagonal (Diagonally dominant-like)
- GB** General Band
- GE** GEneral matrices (e.g., unsymmetric, rectangular, etc.)
- GG** General matrices, Generalized problem
- HE** Complex Hermitian
- OR** Orthogonal Real
- PB** Positive definite Banded (symmetric or Hermitian)
- PO** Positive definite (symmetric or Hermitian)
- PT** Positive definite Tridiagonal (symmetric or Hermitian)
- ST** Symmetric Tridiagonal Real
- SY** SYmmetric
- TR** TRiangular (possibly quasi-triangular)
- TZ** TrapeZoidal
- UN** UNitary complex

● Drivers ZZZ

P $\textcolor{red}{X}$ $\textcolor{yellow}{X}$ $\textcolor{cyan}{Y}$ $\textcolor{magenta}{Y}$ $\textcolor{blue}{Z}$ $\textcolor{green}{Z}$ $\textcolor{red}{Z}$

SL	Linear Equations (SVX^*)
SV	Least Squares
VD	Singular Value
EV	Eigenvalue (EVX^{**})
GVX	Generalized Eigenvalue
	<p>*Estimates condition numbers. *Provides Error Bounds. *Equilibrates System. **Selected Eigenvalues</p>

PxGEMM (mflops)

N=	2000	4000	6000	8000	10000	
Cray	1055	1070	1075			2x2
T3E	3630	4005	4258	4171	4292	4x4
	13456	14287	15419	15858	16755	8x8
IBM	755					2x2
SP2	2514	2850	3040			4x4
	6205	8709	9861	10468	10774	8x8
Now	463	470				2x2
Berkeley	926	1031	632	1754		4x4
	4130	5457	6041	6360	6647	8x8

PxGESV (mflops)

N=	2000	5000	75000	10000	15000	
Cray	702	884	932			1x4
T3E	1508	2680	3218	3356	3602	2x8
	2419	6912	9028	10299	12547	4x16
IBM	421	603				1x4
SP2	722	1543	1903	2149		2x8
	721	2084	2837	3344	3963	4x16
Now	350					1x4
Berkeley	811	1310	1472	1547		2x8
	1171	3091	3937	4263	4560	4x16

Rules of Thumb

- Number of PEs to use = $M \times N / 10^{**6}$.
- Use a square processor grid, $mp = np$.
- Use block size 32 (Cray) or 64
 - for efficient matrix multiply.
- Use vendor's BLAS.
- Bandwidth/mode (MB/sec) >
peak rate (MFLOPS)/**10**
- Latency < **500** usecs.

Example: Linear Equation Solution Ax=b

```
program ples
!    Parallel Linear Equation solver
!    Array A = ( MP*MLA x NP*NLA )
!    Proc. grid is MP   x NP
!    Local mat. (al) MLA x  NLA
!    array A(ig,jg) = 0.1*ig + 0.001*jg (ig not= jg)
!                = ig          (ig  = jg)
!    Solve Ax=b, b=1
integer, parameter :: MP=2,MLA=4,MB=2,NP=2,NLA=4,NB=2
integer, parameter :: M=MLA*MP,      N=NLA*NP
integer, dimension(10):: idescal, idescb
real,  allocatable :: al(:,:), b(:)
integer, allocatable :: ipiv(:)
DATA      NPROW / 2 / , NPCOL / 2 /

CALL SL_INIT( icon, NPROW, NPCOL )
call BLACS_GET(0,0,icon)
call blacs_gridinit( icon,'c', mp, np ) !MP & NP = 2**x
call blacs_gridinfo( icon, mp_ret, np_ret, myrow, mycol)
```

```
write(*,'( "(,2i2,")," on ",i2," x",i2," grid ", &
& "A(" ,i3, " ,",i3,")" )') myrow,mycol,MP,NP,MLA,NLA
if( (MLA/MB)*MB .ne. MLA) stop 'MLA/MB not even'
if( (NLA/NB)*NB .ne. NLA) stop 'NLA/NB not even'

call descinit(idescal, m,n,mb,nb, 0, 0, icon, MLA, info)
call descinit(idescb, m,1,nb, 1, 0, 0, icon, MLA, info)

allocate( al(MLA,NLA), ipiv(MLA+MB), b(MLA) )
call setarray(al,myrow,mycol,4,4)
b=1.0
call PSGESV(N, 1, al, 1,1,idescal, ipiv, &
            b, 1,1,idescb, info)
if(mycol .eq. 0) write(*,'( "(,2i2,"),4(f8.4) )' ) myrow,mycol,b
deallocate( al, ipiv, b )
end program
```

```
subroutine setarray(a,myrow,mycol,lda_x, lda_y )
    ! distribute a matrix aa to the local arrays.
    implicit none
    real :: aa(8,8)
    real :: bb(8,1)
    integer :: lda_x, lda_y
    real :: a(lda_x,lda_y)
    real :: ll,mm,cr,cc
    integer :: ii,jj,i,j,myrow,mycol,pr,pc,h,g
    integer , parameter :: nprow = 2, npcol = 2
    integer, parameter :: n=8,m=8,nb=2,mb=2,rsrc=0,csrc=0
    integer, parameter :: n_b = 1
    do i=1,8
        do j=1,8
            if(i .eq. j)then
                aa(i,j)=i
            else
                aa(i,j)=0.1*i+0.001*j
            endif
        enddo
    enddo
```

```

do i=1,m
    do j = 1,n
        ! finding out which pe gets this i,j element
        cr = real( (i-1)/mb )
        h = rsrc+aint(cr)
        pr = mod( h,nprow)
        cc = real( (j-1)/mb )
        g = csrc+aint(cc)
        pc = mod(g,nprow)
        ! check if on this pe and then set a
        if (myrow ==pr .and. mycol==pc)then
            ! ii,jj coordinates of local array element
            ! ii = x + l*mb
            ! jj = y + m*nb
            ll = real( ( (i-1)/(nprow*mb) ) )
            mm = real( ( (j-1)/(npcol*nb) ) )
            ii = mod(i-1,mb) + 1 + aint(ll)*mb
            jj = mod(j-1,nb) + 1 + aint(mm)*nb
            a(ii,jj) = aa(i,j)
            write(*,12345) i,j,ii,jj,myrow,mycol
        end if
    end do
end do
12345 format("a(",i2," ",i2,") maps to (" ,i2," ",i2,") on processor ",2i3)
end subroutine

```

```
mpxlf90 -lblacs -lblas -L /usr/local/lib  
-lscalapack -lpblas -ltools ex2.f -o darwin
```

Compile

```
#!/bin/csh  
# @ arguments = " "  
# @ input = /dev/null  
# @ output = mycode.$(jobid).out  
# @ error = mycode.$(jobid).error  
# @ initialdir = /rmount/work/tkaiser/scalapack  
# @ executable = run_it  
# @ job_type = parallel  
# @ notify_user = tkaiser@sdsc.edu  
# @ requirements = (Adapter == "hps_user")  
# @ min_processors = 4  
# @ max_processors = 4  
# @ notification = never  
# @ wall_clock_limit = 00:02:00  
# @ class = normal  
# @ queue  
darwin
```

&

Run Script

```

program map_b
    implicit none
    real :: ll,mm,cr,cc
    integer :: ii,jj,i,j,myrow,mycol,pr,pc,h,g
    integer , parameter :: nprow = 2, npcol = 2
    integer, parameter ::n=8,m=8,nb=2,mb=2,rsrc=0,csrc=0
    do myrow=0,1
        do mycol=0,1
            do i = 1, m
                j = 1
                cr = real( (i-1)/mb )
                h = rsrc+aint(cr)
                pr = mod( h,nprow )
                cc = real( (j-1)/mb )
                g = csrc+aint(cc)
                pc = mod(g,nprow)
                if (myrow ==pr .and. mycol==pc)then
                    ll = real( ( (i-1)/(nprow*mb) ) )
                    mm = real( ( (j-1)/(npcol*nb) ) )
                    jj = mod(j-1,nb) + 1 + aint(mm)*nb
                    ii = mod(i-1,mb) + 1 + aint(ll)*mb
                    write(*,12345) i,j,ii,jj,myrow,mycol
                end if
            12345 format("b","",i1,"",i1,"") maps to ("",i1,"",i1,"") on processor ",2i2)
            end do
        end do
    end do
end

```

Processor 0 output

```
0:( 0 0) on 2 x 2 grid AI( 4, 4)
0:a( 1, 1) maps to ( 1, 1) on processor 0 0
0:a( 1, 2) maps to ( 1, 2) on processor 0 0
0:a( 1, 5) maps to ( 1, 3) on processor 0 0
0:a( 1, 6) maps to ( 1, 4) on processor 0 0
0:a( 2, 1) maps to ( 2, 1) on processor 0 0
0:a( 2, 2) maps to ( 2, 2) on processor 0 0
0:a( 2, 5) maps to ( 2, 3) on processor 0 0
0:a( 2, 6) maps to ( 2, 4) on processor 0 0
0:a( 5, 1) maps to ( 3, 1) on processor 0 0
0:a( 5, 2) maps to ( 3, 2) on processor 0 0
0:a( 5, 5) maps to ( 3, 3) on processor 0 0
0:a( 5, 6) maps to ( 3, 4) on processor 0 0
0:a( 6, 1) maps to ( 4, 1) on processor 0 0
0:a( 6, 2) maps to ( 4, 2) on processor 0 0
0:a( 6, 5) maps to ( 4, 3) on processor 0 0
0:a( 6, 6) maps to ( 4, 4) on processor 0 0
0:( 0 0) 0.9321 0.3773 0.0444 0.0074
```

Processor 1 output

1:(1 0) on 2 x 2 grid AI(4, 4)
1:a(3, 1) maps to (1, 1) on processor 1 0
1:a(3, 2) maps to (1, 2) on processor 1 0
1:a(3, 5) maps to (1, 3) on processor 1 0
1:a(3, 6) maps to (1, 4) on processor 1 0
1:a(4, 1) maps to (2, 1) on processor 1 0
1:a(4, 2) maps to (2, 2) on processor 1 0
1:a(4, 5) maps to (2, 3) on processor 1 0
1:a(4, 6) maps to (2, 4) on processor 1 0
1:a(7, 1) maps to (3, 1) on processor 1 0
1:a(7, 2) maps to (3, 2) on processor 1 0
1:a(7, 5) maps to (3, 3) on processor 1 0
1:a(7, 6) maps to (3, 4) on processor 1 0
1:a(8, 1) maps to (4, 1) on processor 1 0
1:a(8, 2) maps to (4, 2) on processor 1 0
1:a(8, 5) maps to (4, 3) on processor 1 0
1:a(8, 6) maps to (4, 4) on processor 1 0
1:(1 0) 0.1924 0.0999 -0.0190 -0.0388

Processor 2 output

2:(0 1) on 2 x 2 grid AI(4, 4)
2:a(1, 3) maps to (1, 1) on processor 0 1
2:a(1, 4) maps to (1, 2) on processor 0 1
2:a(1, 7) maps to (1, 3) on processor 0 1
2:a(1, 8) maps to (1, 4) on processor 0 1
2:a(2, 3) maps to (2, 1) on processor 0 1
2:a(2, 4) maps to (2, 2) on processor 0 1
2:a(2, 7) maps to (2, 3) on processor 0 1
2:a(2, 8) maps to (2, 4) on processor 0 1
2:a(5, 3) maps to (3, 1) on processor 0 1
2:a(5, 4) maps to (3, 2) on processor 0 1
2:a(5, 7) maps to (3, 3) on processor 0 1
2:a(5, 8) maps to (3, 4) on processor 0 1
2:a(6, 3) maps to (4, 1) on processor 0 1
2:a(6, 4) maps to (4, 2) on processor 0 1
2:a(6, 7) maps to (4, 3) on processor 0 1
2:a(6, 8) maps to (4, 4) on processor 0 1

Processor 3 output

3:(1 1) on 2 x 2 grid Al(4, 4)
3:a(3, 3) maps to (1, 1) on processor 1 1
3:a(3, 4) maps to (1, 2) on processor 1 1
3:a(3, 7) maps to (1, 3) on processor 1 1
3:a(3, 8) maps to (1, 4) on processor 1 1
3:a(4, 3) maps to (2, 1) on processor 1 1
3:a(4, 4) maps to (2, 2) on processor 1 1
3:a(4, 7) maps to (2, 3) on processor 1 1
3:a(4, 8) maps to (2, 4) on processor 1 1
3:a(7, 3) maps to (3, 1) on processor 1 1
3:a(7, 4) maps to (3, 2) on processor 1 1
3:a(7, 7) maps to (3, 3) on processor 1 1
3:a(7, 8) maps to (3, 4) on processor 1 1
3:a(8, 3) maps to (4, 1) on processor 1 1
3:a(8, 4) maps to (4, 2) on processor 1 1
3:a(8, 7) maps to (4, 3) on processor 1 1
3:a(8, 8) maps to (4, 4) on processor 1 1

Some final notes:

- We have a web page that generates code for matrix decomposition
- www.npaci.edu/Resources/Batch/scalapack.html
- Real*8 Available on T3e
- Real*4 and Real*8 on SP